Listing of Claims:

**Claim 1:**

1. (Amended Hereby) A method for assigning to and ordered executing of tasks by instruction processors in a multiple instruction processor computer system having at least three levels of memory, said at least three levels being at least two cache levels, a first of which is accessible directly by a single one of said instruction processors, a mid-level memory being a multiprocessor-accessible cache accessible by at least two of said instruction processors, and a third memory level being a main memory, accessible by all of said instruction processors, said method comprising:

selecting a processor-associated switching queue to which to assign a new task,

assigning said new task to said selected switching queue by placing information about said selected switching queue into said new task,

running a one of said instruction processors based upon tasks having information in said one instruction processor's associated switching queue until there are no tasks in ~~a one of said each~~ said one instruction processor's associated switching queue and then,

determining through the use of a selection matrix which other switching queue may be used as a second level switching queue by said one instruction processor, and

inquiring by said one instruction processor of said second level switching queue for a next task that may be available on said second level switching queue.

**Claim 2:**

2. (Original) The method of claim 1 wherein if said second switching queue has a said next task, assigning affinity for that task to said one instruction processor and allowing said one instruction processor to execute said next task.

**Claim 3:**

3. (Original) The method of claim 1 wherein said selection matrix provides a hierarchical selection agenda through which access to a new switching queue is determined when said one instruction processor's associated switching queue has no tasks and wherein said hierarchical selection agenda first selects said second level switching queue from among switching queues associated with instruction processors on a same bus as said one instruction processor.

**Claim 4:**

4. (Original) The method of claim 3 wherein after said hierarchical selection agenda first selects said second level switching queue from among switching queues associated with instruction processors on a same bus as said one instruction processor, if said one instruction processor cannot find a said next task on said second level switching queue, said hierarchical selection agenda then selects a third level switching queue from among switching queues associated with instruction processors that use a shared cache with said one instruction processor.

**Claim 5:**

5. (Original) The method of claim 4 wherein after said hierarchical selection agenda selects said third level switching queue from among switching queues associated with instruction processors on a shared cache shared with said one instruction processor, if said one instruction processor cannot find a said next task on said third level switching queue, said hierarchical selection agenda then selects a fourth level switching queue from among switching queues associated with instruction processors that use a crossbar with said one instruction processor.

**Claim 6:**

6. (Original) The method of claim 5 wherein after said hierarchical selection agenda selects said fourth level switching queue from among switching queues associated with instruction processors on a shared cache shared with said one

instruction processor, if said one instruction processor cannot find a said next task on said fourth level switching queue, said hierarchical selection agenda then selects from among switching queues associated with instruction processors that use another crossbar to access main memory than the one used by said one instruction processor.

**Claim 7:**

7. (Original) The method of claim 1 wherein said information about said switching queue is placed as header information into said new task.

**Claim 8:**

8. (Original) The method of claim 1 further comprising monitoring busyness of an instruction processor to determine whether to proceed to said inquiring step or to idle said instruction processor and only proceeding to said inquiring step where said monitored busyness reaches a threshold value.

**Claim 9:**

9. (Original) The method of claim 8 wherein said monitoring comprises periodically checking the busyness of instruction processors.

**Claim 10:**

10. (Original) The method of claim 8 wherein said monitoring comprises periodically checking the busyness of each of said instruction processors.

**Claim 11:**

11. (Original) The method of claim 10 further comprising evaluating the relative busyness of said one instruction processor to all others of said instruction processors.

**Claim 12:**

12. (Original) The method of claim 8 wherein said threshold value is set

differently for different levels of switching queues.

**Claim 13:**

13. (Original) The method of claim 1 wherein said selecting of a processor-associated switching queue to which to assign a new task is based upon idleness qualities of all available processor-associated switching queues.

**Claim 14:**

14. (Original) The method of claim 1 wherein said selecting of a processor-associated switching queue to which to assign a new task may accomplished using substantially any one of said instruction processors of said multiple instruction processors.

**Claim 15:**

15. (Original) The method of claim 14 wherein said selecting of a processor-associated switching queue to which to assign a new task may accomplished using substantially every one of said instruction processors of said multiple instruction processors, as each of said substantially every one of said instruction processors becomes ready to seek a new task.

**Claim 16:**

16. (Original) The method of claim 1 wherein said selecting of a processor-associated switching queue to which to assign a new task assigns said task to a switching queue used by a plurality of said instruction processors for their processor-associated switching queue.

**Claim 17:**

17. (Amended Hereby) A method for assigning to and ordered executing of tasks by instruction processors in a multiple instruction processor computer system having hierarchical levels of memory, said method comprising:

selecting a processor-associated switching queue to which to assign a new

task, based upon idleness qualities of all available processor associated switching queues,

assigning said new task to said selected switching queue by placing information about said selected switching queue into said new task,

running a one of said instruction processors based upon tasks having information in said one instruction processor's associated switching queue until there are no tasks in ~~a one of said each~~ said one instruction processor's associated switching queue and then,

determining through the use of a selection matrix which other switching queue may be used as a second level switching queue by said one instruction processor, and

inquiring by said one instruction processor of said second level switching queue for a next task that may be available on said second level switching queue.

**Claim 18:**

18. (Original) A dispatcher algorithm for use in a multiple instruction processor computer system having at least three levels of memory, said three levels being at least two cache levels, a first of which is accessible directly by a single one of said instruction processors, a mid-level memory being a multiprocessor-accessible cache accessible by at least two of said instruction processors, and a third memory level being a main memory, accessible by all of said instruction processors, wherein tasks are directed to switching queues for processor assignment on an affinity basis by an executive and said switching queues are maintained and controlled by said dispatcher algorithm, said dispatcher algorithm comprising:

an executable program for assigning affinity of each new task to a one of said instruction processors executing said executable program,

a set of switching queues of substantially the same number as instruction processors wherein one switching queue is associated with said one of said instruction processors and a switching queue is also associated with substantially each other of said instructions processors, said switching queues having code for their operation and a data area wherein said data area is for maintaining a list of

tasks for an instruction processor to accomplish,

a load balancing level matrix for directing said first instruction processor to steal a task from a switching queue associated with another instruction processor in accord with a predetermined mapping within said matrix when said first instruction processor is looking for an additional task.

**Claim 19:**

19. (Original) The apparatus of claim 18 additionally comprising a monitor program for measuring a level of instruction processor busyness/idleness of said one instruction processor, for comparing the level of busyness/idleness to a predetermined threshold, and if the measured level of busyness/idleness exceeds the threshold, for permitting said one instruction processor to use the load balancing level matrix to determine which of said other instruction processor-associated switching queues to seek a new task from.

**Claim 20:**

20. (Original) The apparatus of claim 19 wherein said monitor operates as executable code of the dispatcher algorithm used by an instruction processor when said instruction processor seeks a new task wherein a value related to a current level of busyness of said instruction processor using said monitor executable code is stored in an instruction processor busyness data area.

**Claim 21:**

21. (Original) The apparatus of claim 20 wherein said monitor evaluates the values stored in said instruction processor busyness data area and generates a level of busyness value of the multiprocessor system there-from, and based on a comparison between said current busyness value for this one instruction processor, produces a transfer affinity value, and wherein said monitor compares said transfer affinity value against a threshold level value to determine whether said another

instruction processor will be permitted to steal a task from said one instruction processor-associated switching queue or not.

**Claim 22:**

22. (Original) A dispatcher algorithm for use in a multiple instruction processor computer system having at least three levels of memory, said three levels being at least two cache levels, a first of which is accessible directly by a single one of said instruction processors, a mid-level memory being a multiprocessor-accessible cache accessible by at least two of said instruction processors, and a third memory level being a main memory, accessible by all of said instruction processors, wherein tasks are directed to switching queues for cluster assignment on an affinity basis by an executive and said switching queues are maintained and controlled by said dispatcher algorithm, said dispatcher algorithm comprising:

an executable program for assigning affinity of each new task to a one of said clusters executing said executable program, wherein a cluster is a group of instruction processors,

a set of switching queues of substantially the same number as clusters wherein one switching queue is associated with said one of said clusters and a switching queue is also associated with substantially each other of said clusters, said switching queues having code for their operation and a data area wherein said data area is for maintaining a list of tasks for a cluster to accomplish,

a load balancing level matrix for directing said first cluster to steal a task from a switching queue associated with another cluster in accord with a predetermined mapping within said matrix when said first cluster is looking for an additional task.

**Claim 23:**

23. (Original) A dispatcher algorithm for use in a multiple instruction processor computer system having hierarchical levels of memory, wherein tasks are directed to switching queues for assignment to a processor unit on an affinity basis by an executive and said switching queues are maintained and controlled by said dispatcher algorithm, said dispatcher algorithm comprising:

an executable program for assigning affinity of each new task to a one of said processor units executing said executable program,

a set of switching queues of substantially the same number as processor units wherein one switching queue is associated with one of said processor units and a switching queue is also associated with substantially each other of said processor units, said switching queues having code for their operation and a data area wherein said data area is for maintaining a list of tasks for an processor unit to accomplish,

a load balancing level matrix for enabling said first processor unit to steal a task from a switching queue associated with another processor unit in accord with a predetermined mapping within said matrix when said first processor unit is looking for an additional task.

**Claim 24:**

24. (Original) A multiple instruction processor computer system having a dispatcher algorithm as set forth in claim 23.

**Claim 25:**

25. (Original) A dispatcher algorithm as set forth in claim 23, wherein each said processor unit comprises either a cluster of instruction processors or a single instruction processor.

**Claim 26:**

26. (Original) A dispatcher algorithm as set forth in claim 25, wherein at least one of said processor units is comprised of a different number of instruction processors than at least one other one of said processor units.